Performance Specifications

Arvind
Computer Science & Artificial Intelligence Lab
Massachusetts Institute of Technology

Simple processor pipeline

- Functional behavior is well understood
- Intuition about performance is lacking
  - Should the branch be resolved in the Decode or Execute stage?
  - Should the branch target address be latched before its use?
- Experimentation is required to evaluate design alternatives

We present a design flow that makes such experimentation easy for the designer
Need for Performance Specs

Rules:

\[
F = \{\text{Fetch}\} \quad D = \{\text{DecAdd, DecBz, } \ldots\} \quad E = \{\text{ExeAdd, ExeBzTaken, ExeBzNotTaken, } \ldots\} \quad M = \{\text{MemLd, MemSt, MemWB, } \ldots\} \quad W = \{\text{Wb}\}
\]

- What is the design’s performance / throughput?
- Reference model implies one rule per cycle execution

**Designer’s goal is usually different and based on the application!**

Pipelining via Performance specification

- The designer wants a pipeline which executes one instruction every cycle
- Performance spec for a pipelined processor:

\[
W < M < E < D < F
\]
More Performance Specification

\[
\begin{align*}
F &= \{\text{Fetch} \} \\
D &= \{\text{DecAdd, DecBz, ...} \} \\
E &= \{\text{ExeAdd, ExeBzTaken, ExeBzNotTaken, ...} \} \\
M &= \{\text{MemLd, MemSt, MemWB, ...} \} \\
W &= \{\text{Wb} \}
\end{align*}
\]

We allow the designer to specify performance!

\[
W < M < E < D < F \quad \equiv \quad \text{pipelined}
\]

1) \( W < M < E^* < D < F \) \quad \equiv \quad \text{pipelined except for ExeBzTaken}

2) \( W < M < \text{ExeBzTaken} \)

What do the following mean?

\[
\begin{align*}
F &< D < E < M < W \\
\equiv &\quad \text{unpipelined (assuming buffers start empty)}
\end{align*}
\]

\[
\begin{align*}
W &< W < M < E < E < D < D < F < F
\equiv &\quad \text{two-way superscalar!}
\end{align*}
\]

Synthesis algorithms ensure that performance specs are satisfied and guarantee that functionality is not altered.

Why is functionality maintained?

- A few observations about rule-based systems:
  - Adding a new rule to a system can only introduce new behaviors
  - If the new rule is a derived rule, then it does not add new behaviors

- Composed rules:
  - Given rules:
    \[
    \begin{align*}
    R_a : &\quad \text{when } \pi_a(s) \Rightarrow s := \delta_a(s) \\
    R_b : &\quad \text{when } \pi_b(s) \Rightarrow s := \delta_b(s)
    \end{align*}
    \]
  - The composed rule is a derived rule:
    \[
    R_{a,b} : \quad \text{when } \pi_a(s) \& \pi_b(\delta_a(s)) \Rightarrow s := \delta_b(\delta_a(s))
    \]

March 5, 2008 http://csg.csail.mit.edu/6.375
Scheduling Specifications

rule fetch_and_decode (!stallfunc(instr, bu));
  bu.enq(newIt(instr,rf));
  pc <= predIa;
endrule

rule execAdd
  (it matches tagged EAdd(dst:.rd,src1:.va,src2:.vb));
  rf.upd(rd, va+vb); bu.deq(); endrule

rule execBzTaken(it matches tagged Bz {cond:.cv,addr:.av}
  &&& (cv == 0));
  pc <= av; bu.clear(); endrule

rule execBzNotTaken(it matches tagged Bz {cond:.cv,addr:.av}
  &&& !(cv == 0));
  bu.deq();
endrule

rule execLoad(it matches tagged ELoad{dst:.rd,addr:.av});
  rf.upd(rd, dMem.read(av)); bu.deq(); endrule

rule execStore(it matches tagged EStore{value:.vv,addr:.av});
  dMem.write(av, vv); bu.deq(); endrule

Implications for modules

rule fetch_and_decode (!stallfunc(instr, bu));
  bu.enq(newIt(instr,rf));
  pc <= predIa;
endrule

rule execAdd
  (it matches tagged EAdd(dst:.rd,src1:.va,src2:.vb));
  rf.upd(rd, va+vb); bu.deq(); endrule

\[ \text{execAdd} \prec \text{fetch} \]

- \text{rf}: \text{sub} > \text{upd}
- \text{bu}: \{\text{find, enq}\} > \{\text{first, deq}\}
Branch rules

rule fetch_and_decode (!stallfunc(instr, bu));
  bu.enq(newIt(instr,rf));
  pc <= predIa;
endrule

rule execBzTaken(it matches tagged Bz {cond:.cv,addr:.av}
  &&& (cv == 0));
  pc <= av; bu.clear(); endrule

rule execBzNotTaken(it matches tagged Bz {cond:.cv,addr:.av}
  &&& !(cv == 0));
  bu.deq(); endrule

- execBzTaken < fetch?
  - Should be treated as conflict – give priority to execBzTaken
- execBzNotTaken < fetch
  bu: \{first, deq\} < \{find, enq\}

Load-Store Rules

rule fetch_and_decode (!stallfunc(instr, bu));
  bu.enq(newIt(instr,rf));
  pc <= predIa;
endrule

rule execLoad(it matches tagged ELoad{dst:.rd,addr:.av});
  rf.upd(rd, dMem.read(av)); bu.deq();
endrule

rule execStore(it matches tagged EStore{value:.vv,addr:.av});
  dMem.write(av, vv); bu.deq();
endrule

- execLoad < fetch?
  - Same as execAdd, i.e.,
    rf: upd < sub
    bu: \{first, deq\} < \{find, enq\}
- execStore < fetch?
  bu: \{first, deq\} < \{find, enq\}
Properties Required of Register File & FIFO to meet performance specs

Register File:
- $\text{rf.upd} < \text{rf.sub}$

FIFO
- $\text{bu: \{first, deq\} < \{find, enq\} \Rightarrow}$
  - $\text{bu.first} < \text{bu.find}$
  - $\text{bu.first} < \text{bu.enq}$
  - $\text{bu.deq} < \text{bu.find}$
  - $\text{bu.deq} < \text{bu.enq}$

The good news ...

It is always possible to transform your design to meet desired concurrency and functionality
- Though critical path and hence the clock period may increase
Register Interfaces

\[ \text{read} < \text{write} \quad \text{write} < \text{read} ? \]

- \text{read}' – returns the current state when write is not enabled
- \text{read}' – returns the value being written if write is enabled

Ephemeral History Register (EHR)

\[ \text{read}^0 < \text{write}^0 < \text{read}^1 < \text{write}^1 < \ldots \]

- \text{write}^{i+1} takes precedence over \text{write}^i

[Rosenband MEMOCODE’04]
Transformation for Performance

rule fetch_and_decode (!stallfunc (instr, bu));
  bu.enq (newIt (instr, rf));
  pc <= predIa;
endrule

rule execAdd
  (it matches tagged EAdd(dst:.rd, src1:.va, src2:.vb));
  rf.upd (rd, va + vb); bu.deq ();
endrule

rule execBzTaken (it matches tagged Bz {cond:.cv, addr:.av})
  && (cv == 0));
  pc <= av; bu.clear();
endrule

rule execBzNotTaken (it matches tagged Bz {cond:.cv, addr:.av})
  && ! (cv == 0));
  bu.deq ()
endrule

rule execLoad (it matches tagged ELoad(dst:.rd, addr:.av));
  rf.upd (rd, dMem.read (av)); bu.deq ()
endrule

rule execStore (it matches tagged EStore(value:.vv, addr:.av));
  dMem.write (av, vv); bu.deq ()
endrule

One Element FIFO using EHRs

module mkFIFO1 (FIFO#(t));
  EHRReg#(t) data <- mkEHRReg2U();
  EHRReg#(Bool) full <- mkEHRReg2 (False);
  method Action enq0 (t x) if (!full.read0);
    full.write0 <= True; data.write0 <= x;
  endmethod
  method Action deq0 () if (full.read0);
    full.write0 <= False;
  endmethod
  method t first0 () if (full.read0);
    return (data.read0);
  endmethod
  method Action clear0();
    full.write0 <= False;
  endmethod
endmodule

method Action enq1 (t x) if (!full.read1);
  full.write1 <= True; data.write1 <= x;
endmethod
Experiments in scheduling
Dan Rosenband, ICCAD 2005

What happens if the user specifies:

\[ Wb < Wb < Mem < Mem < Exe < Exe < Dec < Dec < IF < IF \]

No change in rules  a superscalar processor!

Executing 2 instructions per cycle  requires more resources but is functionally equivalent to the original design

4-Stage Processor Results

<table>
<thead>
<tr>
<th>Design</th>
<th>Benchmark (cycles)</th>
<th>Area 10ns (µm²)</th>
<th>Timing 10ns (ns)</th>
<th>Area 2ns (µm²)</th>
<th>Timing 2ns (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 element fifo:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Spec</td>
<td>18525</td>
<td>24762</td>
<td>5.85</td>
<td>26632</td>
<td>2.00</td>
</tr>
<tr>
<td>Spec 1</td>
<td>11115</td>
<td>25094</td>
<td>6.83</td>
<td>33360</td>
<td>2.00</td>
</tr>
<tr>
<td>Spec 2</td>
<td>11115</td>
<td>25264</td>
<td>6.78</td>
<td>34099</td>
<td>2.04</td>
</tr>
<tr>
<td>2 element fifo:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Spec.</td>
<td>18525</td>
<td>32240</td>
<td>7.38</td>
<td>39033</td>
<td>2.00</td>
</tr>
<tr>
<td>Spec 1</td>
<td>11115</td>
<td>32535</td>
<td>8.38</td>
<td>47084</td>
<td>2.63</td>
</tr>
<tr>
<td>Spec 2</td>
<td>7410</td>
<td>45296</td>
<td>9.99</td>
<td>62649</td>
<td>4.72</td>
</tr>
</tbody>
</table>

benchmark: a program containing additions / jumps / loadc's

Dan Rosenband & Arvind 2004
Summary

- For most designs BSV Compiler does good scheduling of rules with some user annotations for priority.
- However, for complex designs sometimes concurrency control is quite difficult and requires a good understanding on the part of the designer of the concurrency issues.
- Performance specification is a good, safe solution but is not implemented in the compiler yet.
  - User can do manual "renaming" and use EHRs to meet most performance goals.
- RWires can solve any problems but exacerbate the correctness issue.
- Synchronous pipelines (single rule) can avoid many problems but is not recommended for complex designs.